**API REST Node Express**

* Initialiser le projet : npm init -y
* Installer les modules : npm i express cors
* Installer nodemon en global et mode dev : npm i –save-dev -g nodemon
* Créer le fichier server
* Importer les modules
* Initialiser l’Api
* Mettre en place le routage (route inexistante)
* Démarrer le serveur
* Changer les script dans package.json avec les deux modes
* Installer dotenv : npm i dotenv
* Créer un fichier .env à la racine (**voir fichier .env**)
* Déclarer les variables d’environnement

Deux façons d’intégrer. env au projet :

* Faire des imports dans chaque fichier qui en a besoin

require ('dotenv').config()

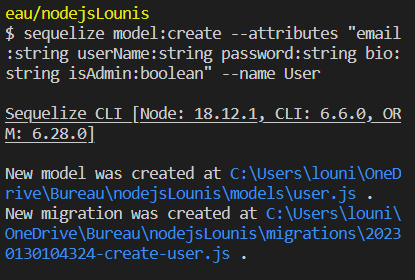
* L’inclure de façon générale dans la ligne de démarrage comme suit :

"scripts": {

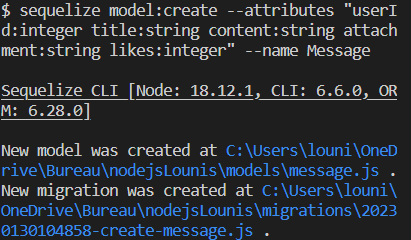
"start": "node -r dotenv/config server.js", // Mode dev

"dev": "nodemon -r dotenv/config server.js"  }, // Mode watcher

* Pour appeler une variable d’environnement : process.env.(nom de la varable). ***Ex. process.SERVER\_PORT***
* Les variables d’environnement sont envoyées en string. Faire un ***parseInt*** si nécessaire.
* Installer sequelize Cli en global : **npm i sequelize-cli**
* L’on peut initialiser l’ORM sequelize : sequelize init ( Cela créera les dossiers/fichiers nécessaires). Ici cela se fera manuellement.
* Dans la ligne de commande création des modèles :



Puis :



* Nouvelle architecture

Une image contenant texte

Description générée automatiquement

* Utiliser l’ORM Sequelize : - Installer le module ***: npm i sequelize mysql2***
* Mysql2 est un module utilisé par sequelize pour communiquer em mysql

**Système de connexion à une base de donnée avec Sequelize:**

* Créer un fichier de connexion (***db.config.js***) (***voir le fichier*** )
* // Dans le module sequelize il y a plein de choses. Ici, on
* // utilise les accolades pour importer uniquement sequelize.
* const {Sequelize}= require ('sequelize');

Tester la connexion à la base de données et lancer le serveur : (***Modifier le fichier server.js)***

/\*\*\* Import de la connexion à la DB  \*/

let DB = require('./db.config')

/\*\*\* Démarrer le serveur avec un test de la DB \*/

DB.authenticate()

    .then (()=> console.log ('Database connection OK!'))

    .then(()=> {

        app.listen (process.env.SERVER\_PORT, ()=> {

        console.log(`The server is running on port ${process.env.SERVER\_PORT}. Have fun !`)

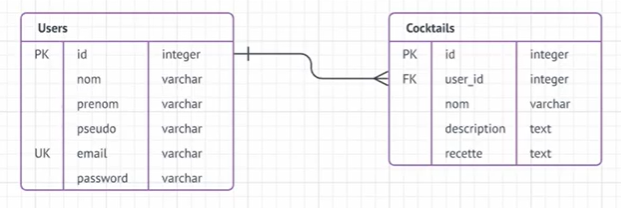
        })

    })

.catch (error => console.log ('Database Error!',error) )

**Création des modèls :**

**La base de données cocktail**



* Créer un dossier models
* Chaque donnée doit être typée dans un modèle ( importer DataTypes de sequelize)

const {DataTypes}= require ('sequelize');

* Importer le fichier de connexion pour le connecter aux modèles

const DB= require ('../db.config');

* Définir le modèle User (***voir le fichier models/user.js*** )
* Définir le modèleCocktail (***voir le fichier models/cocktail.js*** )
* Convention : les noms des modèles commencent par une majuscule (User, Cocktail)

**Le routage des modèles : (***Comment accéder aux ressources***)**

* Créer un dossier routes ;
* Créer un fichier routes/user.js
* Utiliser express pour les routes
* Utiliser le modèle User
* Récupération du router express : **let router= express.Router()** // Attention let et non const
* Créer les routes (ressources) de User (*Voir le fichier*)

**Hachage du mot de passe (** *avec**bcrypt***)**

* Installer bcrypt : npm i bcrypt
* bcrypt.hash (password, parseInt(process.env.BCRYPT\_SALT\_ROUND))
* .then(hash=> {
* req.body.password=hash;
* // Créer le user

User.create(req.body)

* .then (user => res.json({ message:"User created !", data: user }))
* .catch (err => res.status(500).json({message: " Database Error !", error: err}))
* })

})

* .catch(err=> res.status(500).json({message: "Hash Process Error !", error: err}))

**Synchronisation d’un modèle :**

* L’on peut synchroniser chaque modèle à part dans chaque fichier modèle
* // Synchronisation du modèle User
* User.sync((err)=> {
* console.log('Database Sync Error !', err)
* })
* User.sync({force:true});
* User.sync({alter:true});

Ou tous les modèles à la fois dans db.config.js

// Synchronisation des modèles

sequelize.sync((err)=> {

    console.log('Database Sync Error !', err)

})

Authentification JWT:

* Dans le dossier routes, on crée un fichier auth.js pour les routes d’authentification
* Vérifier s'il y'a l'email et le password dans le body de la requête
* Vérifier l'existance de l'email dans la DB
* Vérifier le mot de passe si l’email existe bcrypt.compare(password, user.password)
* Génération du token si le mot de passe est correct : const token = jwt.sign ()
* Exporter le module
* Importer le module dans server.js

**Fermeture de portes (ressources ) :**

* Créer un dossier jsonwebtoken
* Créer un fichier check.js
* Vérifier la présence d’un token
* Extraire le token
* Vérifier la validité du token
* Exporter le middleware et l’utiliser sur des routes à bloquer

**Créer des contrôleurs**

* Créer un dossier conrollers
* Sortir la logique / la mécanique su dossier router et la séparer dans controllers
* Exemple : pour la route auth (login)
* Créer un fichier controllers/auth.js :

exports.login = ((req, res)=> {

* Dans le dossier router :

Importer le fichier controllers/auth.js :

const authCtrl = require ( '../controllers/auth')

Router la route :

router.post('/login', authCtrl.login)

**Gestion des erreurs**

* Utiliser **async await try catch** à la place de **then catch** lorsqu’il y a plusieurs erreurs à gérer

Exemple : Pour afficher un cocktail défini

exports.getOneCocktail = async (req, res)=> {

    let cocktailId= parseInt(req.params.id);

    // Vérifier si le champs id est présent et cohérent

    if(!cocktailId) {

        return res.status(400).json( { message: 'Missing parameter !'})

    }

    try { // Cette fonction risque d’être retardée

        let cocktail = await Cocktail.findOne({ where: {idd:cocktailId}, raw: true})

        // Cocktail pas trouvé

        if(cocktail===null) {

        return res.status(400).json({message: ' This cocktail does not exists !'})

        }

        // Cocktail trouvé

        return res.json({data: cocktail})

    } catch (err) {

        res.status(500).json({ message: 'Database Error !', error: err})

    }

}

* Lorsqu’il y a plusieurs types d’erreurs à gérer par catch, l’on peut faire ainsi :

catch (err) {

        if(err.name==='SequelizeDatabaseError') {

            return res.status(500).json({message: " Database error !", error: err})

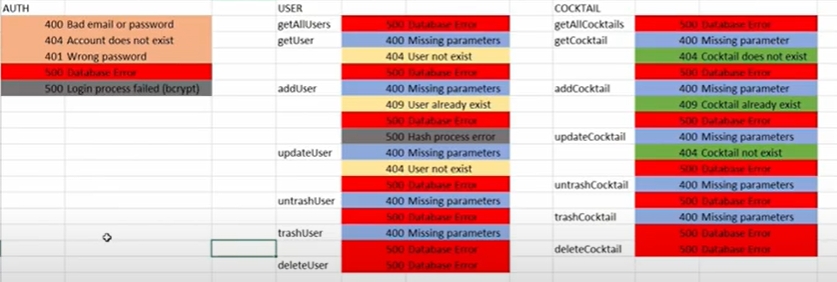
        }

        return res.status(500).json({message: 'login process failed!', error: err})

}

**Organiser le système de gestion des erreurs**

***Les différentes erreurs à gérer***

****

* Créer un fichier errors/customErrors.js (voir fichier)
* Créer un fichier middleware errors/errorHandler.js pour renvoyer les réponses erreurs selon les cas

**Les hooks des modèles Sequelize :**

* Chaque modèle possède des méthodes de hooks à utiliser <https://sequelize.org/docs/v6/other-topics/hooks/>
* Exemple : Mettre le hachage du mot de passe dans le modèle user

// Utilisation du hook befforeCreate pour hacher le mot de passe

User.beforeCreate ( async (user, options) => {

    let hash = await bcrypt.hash (user.password, parseInt(process.env.BCRYPT\_SALT\_ROUND))

    user.password=hash;

})

* Créer une méthode dans le modèle user pour comparer les mots de passe

**LE Cors Policy**

* Les navigateurs web ont un problème de Cors Policy pas Postman qui n’est pas un navigateur
* app.use(cors());